A POLYNOMIAL TIME ALGORITHM FOR SAT

SERGEY GUBIN

Abstract. We define a relation of compatibility on partial true assignments using the system's equations. The relation is a Boolean matrix - the compatibility matrix. We deplete the matrix eliminating relations between those partial true assignments which are not parts of a solution of the whole system. The depletion transforms the relation of compatibility on the set of partial true assignments into the relation of "they are parts of one solution of the system" on the set. The relation is a Boolean matrix - the general solution of a Boolean system. Values "true" in the general solution indicate which partial true assignments can be extrapolated to a full solution and in which "directions" the extrapolation has to be done. We use the method and solve SAT.

Introduction

We describe a method to solve the following system of Boolean equations:

$$g_i(b_{i,1}, b_{i,2}, \ldots, b_{i,n_i}) = h_i(b_{i,1}, b_{i,2}, \ldots, b_{i,n_i}),\ i = 1, 2, \ldots, m \tag{0.1}$$

- where g_i and h_i are given Boolean functions of n_i Boolean variables $b_{i\mu}$, $\mu = 1, 2, \ldots, n_i$. The method has the following three features.

The first feature is an encoding of system 0.1 by a Boolean matrix[1]. We call the matrix a *(true-assignment) compatibility matrix*. The matrix reflects a relation of *compatibility* between true assignments for different equations 0.1: by definition, two such partial true assignments are *compatible* if they do not contradict one another and they both satisfy their equations. In other words, the compatibility matrix encodes system 0.1 in contradictions between the partial true assignments. And we shall emphasize here the words "true assignments."

The second feature is a dynamic programming procedure which iterates the compatibility matrix. We call the procedure *depletion* of the compatibility matrix because, on each iteration, some of the *true*-elements in the compatibility matrix get their value inverted to $false$: we eliminate the relation of compatibility between those partial true assignments which are not parts of the same solution of the whole system. In other words, the depletion transforms the relation of compatibility on the set of partial true assignments into the relation of "they are parts of one solution of the whole Boolean system" on the set.

The third feature is the matrix which emerges after depletion of the compatibility matrix - the final relation. Any *true*-element in this depleted compatibility matrix presents two parts of some true assignment satisfying the whole system 0.1. So, we call this matrix a *general solution* of system 0.1. Substitution of these partial

2000 *Mathematics Subject Classification.* Primary 68Q15, 68R10, 90C57.
Key words and phrases. Boolean system, Algorithm, SAT.
Author's email: sgubin@genesyslab.com.

[1]Boolean matrix is a matrix whose elements are *true* or $false$. Those elements which equal *true* are *true*-elements, and those elements which equal $false$ are $false$-elements.

true assignments in system 0.1 will reduce the system. In other words, the general solution consists of the correct guesses for the method of self-reduction.

Based on that, we classify our method as a search algorithm. Particularly, system 0.1 is consistent iff its general solution contains *true*-elements.

We deploy the method and solve SAT in polynomial time[2].

Readers may try the method on their own examples using demo [4].

1. True-assignment compatibility matrix

For each equation in system 0.1, let's build its truth table as follows:

$\tau_i =$

	True assignments	g_i =? h_i (it is *true* if they are equal, and it is $false$ otherwise)
1	1-st assignments	1-st value of $g_i =?h_i$
2	2-nd assignments	2-nd value of $g_i =?h_i$
$\vdots$	$\vdots$	$\vdots$
2^{n_i}	2^{n_i}-th assignments	2^{n_i}-th value of $g_i =?h_i$

- where n_i is the number of variables in the i-th equation of system 0.1. The true assignments in the tables are ordered arbitrarily.

Let $r_{i\alpha}$ be the α-th row from truth table τ_i, and let $r_{j\beta}$ be a β-th row from truth table τ_j. The rows may be taken from one table ($i = j$), and they even may be the same row ($i = j$ and $\alpha = \beta$). By definition, these rows are *compatible* if they satisfy the following two conditions:

Condition 1: Value of $g_i =?h_i$ and value of $g_j =?h_j$ are *true* in both rows;
Condition 2: True assignments in these rows do not contradict one another, i.e. all common variables in the rows have the same value *true* or $false$ assigned to them.

The relation of compatibility is a symmetric and reflexive relation on the set of all rows in truth tables τ_i, $i = 1, 2, \ldots, m$.

For each couple of equations from system 0.1, let's build a *(true-assignment) compatibility box*. That is the following Boolean matrix:

$$T_{ij} = (t_{ij\mu\nu})_{2^{n_i}\times 2^{n_j}},\ t_{ij\mu\nu} = \begin{cases} true, & \text{Rows } r_{i\alpha} \text{ and } r_{j\beta} \text{ are compatible} \\ false, & \text{Otherwise} \end{cases}$$

Box T_{ij} presents the relation of compatibility between true assignments for i-th and j-th equations in system 0.1. Because the relation is symmetric and reflexive, the boxes have the following symmetries:

$$\begin{aligned} & T_{ji} = T_{ij}^T \\ & T_{ii} = \operatorname{diag}(t_{i,i,1,1}\ t_{i,i,2,2}\ \ldots\ t_{i,i,2^{n_i},2^{n_i}})_{2^{n_i}\times 2^{n_i}} \end{aligned} \tag{1.1}$$

- where operator "$*^T$" is matrix-transponation; and "diag" means a diagonal matrix, i.e. a matrix in which all off-diagonal elements equal $false$.

Let's aggregate all compatibility boxes T_{ij}, $i, j = 1, 2, \ldots, m$, in a box matrix in accordance with their indexes[3]:

$$T = (T_{ij})_{m\times m},\ T_{ij} = (t_{ij\mu\nu})_{2^{n_i}\times 2^{n_j}}$$

[2]See Appendix for explanation of SAT.

[3]We might aggregate the boxes in any other consistent way.

- where size of matrix T is shown in boxes. In elements, matrix T has size

$$(\sum_{i=1}^{m} 2^{n_i}) \times (\sum_{i=1}^{m} 2^{n_i}) = O(2^n m \times 2^n m)$$

- where n_i were defined in 0.1, and

$$n = \max_{1 \le i \le m} n_i \tag{1.2}$$

Due to symmetries 1.1, matrix T is a symmetric matrix.

We call matrix T a *(true-assignment) compatibility matrix* for system 0.1. The matrix encodes the system in contradictions between true assignments for separate equations in the system[4].

Any true assignment satisfying the whole system 0.1 creates in its compatibility matrix a grid of *true*-elements, one element per compatibility box:

$$\{t_{ij\mu\nu} = true \mid \mu = \mu(i),\ \nu = \nu(j)\}$$

- where index μ is a function of index i, and index ν is a function of index j. Based on that, we call any grid of *true*-elements in the compatibility matrix, one element per compatibility box, a *solution grid*.

Due to symmetries 1.1, the index functions $\mu()$ and $\nu()$ in any solution grid are the same function:

$$\mu() = \nu() = \gamma() : \ \{1, 2, \dots, m\} \ \rightarrow \ \{1, 2, \dots, m\}$$

So, specking in indexes, the solution grid is just such function $\gamma()$ that

$$t_{i,j,\gamma(i),\gamma(j)} \equiv true, \ i, j = 1, 2, \dots, m \tag{1.3}$$

Theorem 1.1. *System 0.1 is consistent iff its true-assignment compatibility matrix contains solution grids.*

Proof. If system 0.1 is consistent, then its compatibility matrix has solution grids: any true assignment satisfying the whole system creates a satisfying true assignments for each of the system's equations. In the equations' truth tables, rows appropriate to these satisfying true assignments are compatible. Thus, they create a grid of *true*-elements in compatibility matrix T, one element per compatibility box.

On the other hand, let the compatibility matrix have a solution grid. In the truth tables, rows appropriate to the grid's elements do not contradict each other and the partial true assignment in each of the rows satisfies its equation. Thus, we can "glue" these partial true assignments over their common variables. The result will be a true assignment satisfying the whole system 0.1. □

Theorem 1.1 reduces the solution of system 0.1 to a search for solution grids in the system's compatibility matrix. The matrix is a function of four integer arguments:

$$\tau : \ (i, j, \mu, \nu) \ \mapsto \ t_{ij\mu\nu} \in \{false, true\}$$

- where $t_{ij\mu\nu}$ is the (μ, ν)-th element in the (i, j)-th compatibility box of matrix T. Thus, the search for solution grids is solution of the following functional equation:

$$\tau(i, j, \gamma(i), \gamma(j)) = true \tag{1.4}$$

[4]Obviously, we might use matrix $\neg T$ instead of matrix T. Then, we would swap all operations "$\wedge$"/"$\vee$" and all constants $true/false$.

- where $\gamma()$ is the unknown function of one integer argument. Any solution of equation 1.4 delivers solution grid 1.3, and visa versa.

We will solve equation 1.4 using the method of dynamic programming: we will iteratively replace in the compatibility matrix all values *true* by values $false$ except the solution grids' elements. We call the procedure *depletion* of the compatibility matrix. Due to theorem 1.1, the matrix emerging after the depletion will be a relation of "they are parts of one solution of the whole system," i.e. the matrix will encode all solutions of system 0.1 and only them. In other words, the matrix will be the system's *general solution.*

2. Depletion

Depletion of the compatibility matrix for system 0.1 can be expressed by the following *asynchronous algorithm*[5]:

Iteration: For index triplet (i, μ, j), recalculate compatibility box T_{ij} as follows[6]:

$$T_{ij} = T_{ij} \wedge T_{i\mu}T_{\mu j} \tag{2.1}$$

- where all indexes are in their ranges.

Due to symmetries 1.1, formula 2.1 may be expanded in either way:

$$t_{ij\alpha\beta} = t_{ij\alpha\beta} \wedge \bigvee_{\nu=1}^{2^{n_\mu}} t_{i\mu\alpha\nu} \wedge t_{\mu j\nu\beta} = t_{ij\alpha\beta} \wedge \bigvee_{\nu=1}^{2^{n_\mu}} t_{i\mu\alpha\nu} \wedge t_{j\mu\beta\nu} = \dots$$

- where all indexes are in their ranges.

Iterations: Select an *iteration schema* to tour all index triplets

$$\{(i, \mu, j) \mid i, \mu, j = 1, 2, \dots, m\} \tag{2.2}$$

Cycle index triplets 2.2 in accordance with the schema and repeat depletions 2.1 for each visited index triplet until all compatibility boxes T_{ij} get finalized, i.e. until compatibility matrix T will stop to change.

Due to the conjunction in formula 2.1, the iterations can invert only *true*-elements in matrix T. The matrix has $O(2^{2n}m^2)$ elements in total. Thus, it will get finalized eventually and the procedure will stop.

Actually, the depletion procedure defines a family of algorithms which differ one from another by their iteration schemes. But, regardless of the schemes, all of them preserve only those *true*-elements which belong to solution grids[7].

[5]The algorithm depletes the compatibility boxes asynchronously.

[6]Here, product of Boolean matrices $A = (a_{ij})$ and $B = (b_{ij})$ of the appropriate sizes is the following matrix:

$$AB = (\bigvee_{\nu} a_{i\nu} \wedge b_{\nu j})$$

Conjunction of Boolean matrices of the same size is the matrix of conjunctions of the appropriate elements. And formula 2.1 may be seen as a "square" of matrix T.

[7]Based on author's correspondence, it shall be emphasized that we deal not with compatibility of equations, but with the compatibility of partial true assignments, which was defined through the equations. For clarity, let's assume that the equations are clauses in a CNF. Then, one might abuse the metaphor "contradiction between clauses" and purport "counter example": an unsatisfiable CNF in which every three clauses are satisfiable. Well, let's go a little deeper: because clauses share common variables, their true assignments are intertwined; so, if (the same) one partial true assignment satisfies every three clauses, then it satisfies the whole CNF, indeed, no matter what the rest variables are (the asynchronous algorithm exploits this "globalism" of the partial true

Theorem 2.1. *In the true assignments compatibility matrix, elements belonging to solution grids are the only invariants under depletions 2.1.*

Proof. Direct calculations show that any solution grid is an invariant under iterations/depletions 2.1. So, we just need to show that any $true$-element in that final matrix emerging from the depletions belongs to a solution grid. For that, we will use mathematical induction over m - the number of equations in system 0.1.

In the case of $m = 1$, compatibility matrix consists of one compatibility box only:

$$T = (T_{1,1})_{1\times 1}$$

Due to symmetries 1.1, box $T_{1,1}$ is a diagonal matrix. It has $true$-elements on its diagonal iff the only equation in the system has solutions:

$$g_1(b_{1,1}, b_{1,2}, \ldots, b_{1,n_1}) = h_1(b_{1,1}, b_{1,2}, \ldots, b_{1,n_1})$$

On the other hand, solution grids in matrix T for this case of $m = 1$ are its diagonal $true$-elements. So, the theorem is true in this case.

Now, let $m > 1$ and let's assume that the theorem is true for all systems 0.1 with $m - 1$ equations.

Let's select from the whole mess of depletions 2.1 those depletions which involve the compatibility boxes for the first $m-1$ equations of system 0.1. Those equations are

$$g_1 = h_1,\ g_2 = h_2,\ \ldots,\ g_{m-1} = h_{m-1} \tag{2.3}$$

Due to our induction hypothesis, these depletions will produce the general solution for these first $m-1$ equations. Due to the conjunctions in depletions 2.1, any $false$-element in this restricted general solution will be a $false$-element in the general solution for the whole system 0.1 as well. So, the values of the compatibility boxes for these first $m - 1$ equations, which were produced by the depletion restricted to these first $m - 1$ equations, are the "upper bounds" for the values of the boxes in the general solution for whole system 0.1. Let's use these upper bounds for the values of boxes T_{ij}, $i, j = 1, 2, \ldots, m - 1$.

Let's suppose that these first $m - 1$ equations are inconsistent. Then, system 0.1 is inconsistent as well, and our algorithm has to produce the general solution entirely filled with value $false$. And it really happens: due to theorem 1.1, there is not any solution grid in the compatibility matrix restricted to the first $m - 1$ equations; then, due to our induction hypothesis, the restricted general solution is filled with value $false$ entirely; then, depletions 2.1 will propagate this values $false$ all over the general solution for the whole system 0.1. Thus, the theorem is true in this case.

Let's suppose that the first $m - 1$ equations are consistent. Then, due to our induction hypothesis, the general solution for these first $m-1$ equations consists of their solution grids, only. Still, there are two possibilities: either the whole system 0.1 is consistent or it is inconsistent. Let's analyze these possibilities together.

assignments); thus, in the case of unsatisfiable CNF, for any given partial true assignment, no matter how many clauses it satisfies, there always is such clause whose satisfying true assignments contradict with that given one (see how we arrive at inequality 2.5 in the proof of theorem 2.1); so, iterations 2.1 will make the compatibility matrix's elements equal $false$ one by one. Interested readers may enter the CNF in demo [4] and see how the algorithm will deal with it.

Let the α-th true assignment for the m-th equation

$$g_m = h_m \tag{2.4}$$

contradict all true assignments satisfying the first $m-1$ equations 2.3. Then, there is at least one equation among equations 2.3 which is not satisfied by this α-th true assignment for equation 2.4. Let it be the i_0-th equation, $1 \le i_0 \le m-1$:

$$g_{i_0}(b_{i_0,1}, b_{i_0,2}, \ldots, b_{i_0,n_{i_0}})|_\alpha \ne h_{i_0}(b_{i_0,1}, b_{i_0,2}, \ldots, b_{i_0,n_{i_0}})|_\alpha \tag{2.5}$$

- where the variables have a true assignment appropriate to the α-th true assignment for equation 2.4.

Then, all *true*-elements in the α-th column in compatibility box $T_{i_0 m}$ and all *true*-elements in diagonal compatibility box $T_{i_0 i_0}$ are located in different rows:

$$\forall\ \mu, \nu = 1, 2, \ldots, n_{i_0}\ (t_{i_0 i_0 \mu\nu} \wedge t_{i_0 m \mu\alpha} = false)$$

Then, because matrix $T_{i_0 i_0}$ is a diagonal matrix, depletion

$$T_{i_0 m} = T_{i_0 m} \wedge T_{i_0 i_0} T_{i_0 m}$$

will fill up this α-th column in compatibility box $T_{i_0 m}$ with value $false$ entirely:

$$\forall\ \nu = 1, 2, \ldots, 2^{n_{i_0}}\ (t_{i_0 m \nu\alpha} = t_{i_0 m \nu\alpha} \wedge \bigvee_{\mu=1}^{2^{n_{i_0}}} \underbrace{(t_{i_0 i_0 \mu\nu} \wedge t_{i_0 m \mu\alpha})}_{=false} = false)$$

Then, the next iterations 2.1 will propagate this value $false$ from that α-th column in compatibility box $T_{i_0 m}$ all over the whole α-th column in the m-th box column in compatibility matrix T and further. Thus, due to symmetries 1.1, in the general solution for system 0.1 there will be the following:

$$\forall\ i = 1, 2, \ldots, m,\ \mu = 1, 2, \ldots, n_i\ (t_{im\mu\alpha} = t_{mi\alpha\mu} = false)$$

Particularly,

$$t_{mm\alpha\alpha} = false \tag{2.6}$$

Thus, for any *true*-element $t_{im\mu\alpha}$ in the m-th box column in the final matrix T, the α-th true assignment for equation 2.4 has to be consistent with at least one solution grid for the first $m-1$ equations 2.3. But, in this case, system 0.1 will be consistent and this *true*-element will belong to a solution grid in the system's compatibility matrix. Thus, the theorem is true in this case too. □

Theorem 2.2. *System 0.1 is consistent iff there are true-elements in its general solution.*

Proof. This theorem is a corollary of theorems 1.1 and 2.1: system 0.1 is consistent iff there are solution grids in its compatibility matrix; solution grids are the only invariants under depletions 2.1. □

Theorem 2.2 shows that the general solution of inconsistent system 0.1 is filled with value $false$ entirely. Usually, this filling up happens gradually[8]: at first, a few compatibility boxes become filled with $false$; and then, depletions 2.1 propagate this value $false$ all over the compatibility matrix. We call these first $false$-boxes a *pattern of unsatisfiability*. Due to the conjunctions in depletions 2.1, the algorithm may be stopped prematurely when the pattern arises.

[8]Readers may see it using demo [4]

Theorem 2.3. *Computational complexity of the asynchronous algorithm for system 0.1 is $O(2^{3n}m^3)$, where m is the number of equations in system 0.1 and n is maximum 1.2.*

Proof. For each of the $O(m^3)$ different index triplets (i, μ, j), let's select those depletions 2.1 which involve the compatibility boxes with these indexes (due to symmetry of the compatibility matrix, there are only three really different boxes with these indexes):

$$\begin{array}{lll} T_{ij} = T_{ij} \wedge T_{i\mu}T_{\mu j} & T_{i\mu} = T_{i\mu} \wedge T_{ij}T_{j\mu} & T_{j\mu} = T_{j\mu} \wedge T_{ji}T_{i\mu} \\ T_{ji} = T_{ji} \wedge T_{j\mu}T_{\mu i} & T_{\mu i} = T_{\mu i} \wedge T_{\mu j}T_{ji} & T_{\mu j} = T_{\mu j} \wedge T_{\mu i}T_{ij} \end{array}$$

The boxes have size $O(2^n \times 2^n)$. So, they will deplete each other with $O(2^{3n})$ Boolean operations "$\wedge$" and "$\vee$". □

Theorems 1.1 through 2.3 solve Boolean system 0.1 in time cubical over the number of equations in the system and the maximal number of rows in the equations' truth tables.

From a practical perspective, let's notice that there was no need to account for those partial true assignments which do not satisfy their equations because these partial true assignments created the rows/columns completely filled with $false$ from the very beginning. Thus, these true assignments might be missed during construction of the compatibility boxes, and all of the above might be applied only to those partial true assignments which satisfy their equations. So, the method exploits the contradictions between partial true assignments satisfying different equations 0.1 separately.

3. SAT

Let f be a CNF:

$$f = c_1 \wedge c_2 \wedge \ldots \wedge c_m \tag{3.1}$$

- where c_i, $i = 1, 2, \ldots, m$, are clauses. Let

$$K = \max_{1 \le i \le m} k_i \tag{3.2}$$

- where k_i is the number of literals in clause c_i.

For SAT instance defined by formula 3.1, we may replace the formula by the following Boolean system:

$$c_i = true, \; i = 1, 2, \ldots, m \tag{3.3}$$

Then, we can compute the system's general solution. With that, we will be able to decide whether formula f is satisfiable as well as find particular satisfying true assignments for the formula. Also, we can use the general solution to count the formula's satisfying true assignments (different solution grids).

Because formula 3.1 is a CNF,

$$n = K$$

- where n is maximum 1.2 and K is maximum 3.2. Thus, the general solution method will resolve formula 3.1 in time $O(2^{3K}m^3)$. Particularly, it will be a cubic time algorithm for 1SAT, 2SAT, 3SAT, etc.

Let S be the *resolution function* for SAT:

$$S(f) = \begin{cases} true, & f \text{ is satisfiable} \\ false, & f \text{ is unsatisfiable} \end{cases}$$

- where f is formula 3.1.

Now, let R be Cook's reduction of SAT to 3SAT [1] (computational complexity of the reduction is $O(Km)$, and its result is a 3SAT instance with $O(Km)$ clauses), let T be our encoding (for 3SAT, its computational complexity is quadratic over the number of clauses), and let D be our depletion. It shall be emphasized that all these operations are constructive and their result is the general solution of system 3.3 for 3SAT. The total computational complexity of superposition $D \circ T \circ R()$ is

$$O(Km) + O(K^2m^2) + O(K^3m^3) = O(K^3m^3)$$

Thus, due to equality 2.6, the following $O(K^3m^3)$-time formula explicitly expresses the resolution function for SAT:

$$S(f) = \bigvee_{\mu=1}^{O(Km)} s_{\mu\mu},\ (s_{\mu\nu})_{O(Km \times Km)} = D \circ T \circ R(f) \tag{3.4}$$

- where f is formula 3.1, and Boolean matrix $(s_{\mu\nu})_{O(Km \times Km)}$ is the general solution of system 3.3 for that 3SAT instance to which the given SAT instance was reduced using Cook's reduction.

Ultimately, explicit expression 3.4 of the resolution function reduces SAT to the following Horn-1SAT instance:

$$\bar{b} \wedge (b \vee S(f))$$

- where b is an independent Boolean variable and f is formula 3.1.

4. Lexicographical approach to SAT

Ultimately, we might use distributive laws and rewrite formula 3.1 in disjunctive form (DF):

$$f = \bigvee_{\{(\alpha_1, \alpha_2, \dots \alpha_m) | 1 \le \alpha_p \le k_p\}} Q_{1,\alpha_1} \wedge Q_{2,\alpha_2} \wedge \dots \wedge Q_{m,\alpha_m} \tag{4.1}$$

- where $Q_{i\mu}$ is the μ-th literal in clause c_i.

Obviously, formula f is satisfiable iff there are implicants in the above DF, i.e. there are conjunctions without complementary literals:

$$\exists(\alpha_1, \alpha_2, \dots \alpha_m) :\ Q_{p,\alpha_p} \neq \neg Q_{q,\alpha_q},\ p, q = 1, 2, \dots, m$$

So, we could decide about satisfiability of formula f by searching for the implicants. The only drawback of this approach is the exponential number of the conjunctions in the DF. In total, there are $O(K^m)$ conjunctions in the form. Nevertheless, we can explore the conditions when the implicants exist.

Assuming that clauses in formula 3.1 and literals in the clauses are enumerated, let's compose for each clause couple (c_p, c_q) its *(lexicographical) compatibility box*

$$L_{pq} = (u_{pq\alpha\beta})_{k_p \times k_q}$$

- where k_p is the number of literals in clause c_p, k_q is the number of literals in clause c_q, and elements $u_{pq\alpha\beta}$ are defined as follows:

$$u_{pq\alpha\beta} = \begin{cases} true, & p = q\ \wedge\ \alpha = \beta \\ false, & p = q\ \wedge\ \alpha \neq \beta \\ true, & p \neq q\ \wedge\ Q_{p\alpha} \neq \neg Q_{q\beta} \\ false, & p \neq q\ \wedge\ Q_{p\alpha} = \neg Q_{q\beta} \end{cases}$$

Let's aggregate these compatibility boxes in box matrix L in accordance with their indexes:

$$L = (L_{pq})_{m\times m},\ L_{pq} = (u_{pq\alpha\beta})_{k_p\times k_q}$$

- where size of L is shown in boxes. In elements, this matrix has size

$$(\sum_{p=1}^{m} k_p) \times (\sum_{p=1}^{m} k_p) = O(Km \times Km)$$

- where K is the maximum 3.2.

Matrix L encodes formula 3.1 in lexicographical contradictions between clauses. We call this matrix a *(lexicographical) compatibility matrix* of the formula.

Among properties of the compatibility matrix, let's notice its symmetry:

$$\begin{array}{l} L^T_{pq} = L_{qp} \ \Rightarrow\ L^T = L \\ L_{pp} = \mathrm{diag}(true, true, \dots, true)_{k_p\times k_p} \end{array} \tag{4.2}$$

- due to the above definition of the compatibility boxes.

Let's define *solution grid* in the lexicographical compatibility matrix as a grid of *true*-elements, one element per compatibility box:

$$\{u_{pq\alpha\beta} = true \mid \alpha = \alpha(p),\ \beta = \beta(q),\ p, q = 1, 2, \dots, m\}$$

- where index α depends only on index p, and index β depends only on index q. Due to symmetries 4.2, the index functions are the same function $\gamma()$:

$$\alpha() = \beta() = \gamma() :\ \{1, 2, \dots, m\} \ \rightarrow\ \{1, 2, \dots, m\} \tag{4.3}$$

So, in index terms, solution grid is any such function $\gamma()$ that

$$u_{pq\gamma(p)\gamma(q)} \equiv true$$

- where all indexes are in their ranges.

Theorem 4.1. *Formula 3.1 is satisfiable iff there are solution grids in the formula's lexicographical compatibility matrix.*

Proof. Due to the compatibility boxes' definition, each solution grid $\gamma()$ in matrix L delivers the following implicant in the formula's DF 4.1:

$$Q_{1,\gamma(1)} \wedge Q_{2,\gamma(2)} \wedge \dots \wedge Q_{m\gamma(m)}$$

And visa versa, any implicant in DF 4.1

$$Q_{1,\alpha_1} \wedge Q_{2,\alpha_2} \wedge \dots \wedge Q_{m\alpha_m}$$

delivers the following solution grid in matrix L:

$$\gamma(p) = \alpha_p,\ p = 1, 2, \dots, m$$

□

Theorem 4.1 reduces the search for implicants in DF 4.1 to a search for solution grids in the lexicographical compatibility matrix of formula 3.1. But, unlike the true-assignment compatibility matrix, we cannot use the asynchronous algorithm to search for solution grids in the lexicographical compatibility matrix.

To see it, we might repeat the proof of theorem 2.1 for the lexicographical compatibility matrix. We would find out that we cannot arrive to inequality 2.5 in this case, and there is not any obvious analog of that inequality.

Still, the asynchronous algorithm will preserve the solution grids in the lexicographical compatibility matrix but, except the special CNF, it will preserve practically all *true*-elements in the matrix. So, it is just unpractical for the approach. Yet, here are some workarounds.

For example, we could express the solution grids by the following Boolean equation:

$$\bigwedge_{p=1}^{m} \bigoplus_{\mu=1}^{k_p} \xi_{p\mu} \wedge \bigwedge_{u_{pq\mu\nu}=false} \bar{\xi}_{p\mu} \vee \bar{\xi}_{q\nu} = true \tag{4.4}$$

- where operator "$\bigoplus$" is XOR[9], and the unknowns $\xi_{p\mu}$ are the following indicators:

$$\xi_{p\mu} = \begin{cases} true, & \text{Literal } Q_{p\mu} \text{ enters in an implicant} \\ false, & \text{Otherwise} \end{cases}$$

There is only $O(K)$ satisfying true assignments for each XOR-clause in equation 4.4, and the total number of clauses in the equation is $O(K^2m^2)$. Thus, the general solution method will solve this equation in time $O(K^9m^6)$.

Conclusion

We have described an algorithm to solve systems of Boolean equations. Computational complexity of the algorithm is cubical over the number of equations and the size of their truth tables.

We divided our guesses (true assignments) into the parts appropriate to the system's equations. The compatibility matrix encoded the system in the contradictions between those parts of the guesses. Depletion was a parallel testing of all guesses organized as a search for matches among the parts.

Unlike the search for the "best", this search for the "match" makes the prospects not "compete" but "cooperate", so they do not stall one another and the searching proceeds to a definitive result.

The parallel testing's result was the general solution of the system. The general solution shows which/how parts of the guesses can be glued to extrapolate them into a solution of the whole system. In the case of an inconsistent system, the general solution is completely filled with value *false* (there is nothing to extrapolate).

We pared the general solution method with Cook's reduction of SAT to 3SAT and resolved SAT in cubic time.

We compared the true-assignment approach to SAT with the lexicographical approach to SAT (guesses are conjunctions of literals taken one from each clause). The last seems to be less efficient. The probable cause of that is, in terms of the guess partition, that the lexicographical guesses' parts are more loose than their true-assignment counterparts. If we assume that the true-assignments' parts intertwine/overlap maximally, then the lexicographical parts (the literals) do not intertwine at all, i.e. they overlap minimally. That gives a direction for future research of the lexicographical approach.

We presented web demo [4], which can be freely used to try/test the method.

[9]XOR of several arguments equals *true* if there is only one argument equal *true* among its arguments:

$$x \oplus y = (x \vee y) \wedge (\bar{x} \vee \bar{y})$$
$$x \oplus y \oplus z = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{y} \vee \bar{z})$$

- etc.

Appendix: Explanation of SAT

SAT is the satisfiability problem for conjunctive normal forms (CNF).

Let f be a CNF:

$$f = c_1 \wedge c_2 \wedge \ldots \wedge c_m,\ m \geq 1$$

- where c_i are clauses. The clauses are disjunctions of literals:

$$c_i = L_{i,1} \vee L_{i,2} \vee \ldots \vee L_{i,k_i},\ k_i \geq 1$$

- where L_{ij} are the literals. The literals are Boolean variables or their negations:

$$L_{ij} \in \{b_\mu, \bar{b}_\mu \mid \mu = 1, 2, \ldots, l\},\ l \geq 1$$

- where b_μ are the Boolean variables. Variables b_ν are independent variables.

Normality of conjunctive form f means that (1) all clauses in the form are different and (2) all literals in each clause depend on different variables:

$$\begin{array}{ll} (1) & i_1 \neq i_2 \ \Rightarrow\ c_{i_1} \neq c_{i_2} \\ (2) & \forall i\ (j_1 \neq j_2 \ \Rightarrow\ L_{ij_1} \neq L_{ij_2} \ \wedge\ L_{ij_1} \neq \neg L_{ij_2}) \end{array}$$

- where all indexes are in their ranges.

The size of CNF is measured by the number of Boolean variables involved, the number of clauses, and the maximal number of literals in the clauses:

$$l,\ m,\ K = \max_{1 \leq i \leq m} k_i$$

Obviously,

$$l = O(Km)$$

Satisfiability problem for formula f is the existence problem for such true assignment which will satisfy the formula:

$$true = f(b_1, b_2, \ldots, b_l)|_{b_\mu = \tau_\mu \in \{false, true\}}$$

Along with this decision problem (whether or not the satisfying true assignments exist), there are other problems: a search problem (to find a satisfying true assignment), a counting problem (to count the satisfying true assignments), etc.

Obviously we could use brute force to resolve SAT instances. But, there is a pitfall: in worst case, the number of tests to perform will be 2^l - the task is infeasible for today's computers even in the case of modest l. So, the actual problem is to find feasible methods to resolve SAT instances.

SAT was among the very first NP-complete problems ever discovered. In 1971, Cook [1] proved that polynomial time non-deterministic Turing machine (NDTM) can be expressed by SAT. In other words, SAT is a uniform computational model for a vast class of problems solvable by NDTM in polynomial time. In 1972, based on Cook's theorem, Karp [2] defined NP-problems as problems for which there is a polynomial time many-one reduction to SAT; if there is a polynomial time many-one reduction of SAT to a NP-problem, then that problem is NP-complete, i.e. it too can serve as a uniform model for all NP-problems. Independently, in 1973, Levin [3] proved that SAT is a universal search problem. So now, the result is known as Cook-Levin theorem: SAT is a NP-complete problem.

From a practical perspective, the NP-completeness of SAT means that any efficient algorithm for SAT pared with the appropriate many-one reduction will solve any NP-problem in polynomial time. Here, we describe an efficient algorithm for SAT, i.e. the algorithm in which the number of "elementary" operations is bounded from the top by a function polynomial over sizes of CNF.

References

[1] Stephen Cook, *The complexity of theorem-proving procedures*, In Conference Record of Third Annual ACM Symposium on Theory of Computing, pp.151-158, 1971

[2] Richard M. Karp, *Reducibility Among Combinatorial Problems*, In Complexity of Computer Computations, Proc. Sympos. IBM, Thomas J. Watson Res. Center, Yorktown Heights, N.Y. New York: Plenum, pp. 85-103, 1972

[3] Levin, Leonid, *Universal'nye perebornye zadachi*, Problemy Peredachi Informatsii 9 (3): pp. 265-266, 1973

[4] *3SAT Solver*, Polynomial times, http://www.timescube.com